# AUTONOMOUS SWARMING UNMANNED AERIAL VEHICLES FOR MULTIPLE PERSPECTIVE OBSERVATION

An Undergraduate Research Scholars Thesis

by

TYLER BRYANT

Submitted to the Undergraduate Research Scholars program
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Research Advisor: Dr. Raktim Bhattacharya

May 2016

Major: Aerospace Engineering

# TABLE OF CONTENTS

# ABSTRACT

Autonomous Swarming Unmanned Aerial Vehicles for Multiple Perspective Observation

Tyler Bryant
Department of Aerospace Engineering
Texas A&M University

Research Advisor: Dr. Raktim Bhattacharya
Department of Aerospace Engineering

There is much research being done in the field of swarming robotics and more specifically swarming unmanned aerial vehicles or UAVs. The most common platform used is the quadcopter because of its versatility. The problem to be solved is how to have autonomous swarming quadcopters that can orient themselves towards a point in space while maintaining their position relative to one another and there position relative to the point. It would be necessary to be able to send a command to these UAVs so they can change their pattern around this point in real time while avoiding each other. These quadcopters would also have video recording capabilities in order to observe the point. If this problem was solved, quadcopters could be used for tasks such as recording shots for movie directors, recording shots for athletic events, or used for surveillance. One day these quadcopters could be used to improve people's lives and even their safety. These quadcopters could be sent into a burning building to cooperatively search for a firefighter or to search for someone missing in a forest or to monitor the border. This is what I would like to achieve.

# NOMENCLATURE

| | |
|---|---|
| DOF | Degree of Freedom |
| LHP | Left Hand Plane |
| UAV | Unmanned Aerial Vehicle |
| PID | Proportional Integral Derivative |

# CHAPTER I

# INTRODUCTION

A quadcopter is a popular unmanned aerial vehicle (UAV) platform. The quadcopter has four motors usually spaced out equally. There are six degrees of freedom in which three are rotational and three are translational. With these motors the quadcopter can control its yaw, pitch, and roll. The dynamics of the quadcopter will be discussed in this chapter. Quadcopters are inherently unstable which means that a feedback control system is required. A proportional-integral-derivative (PID) controller will be used for this project and the method in which it is implemented will be discussed.

## Mathematical Model of a Quadcopter

In order to derive the dynamics of the quadcopter, two frames will be chosen. The inertial frame is fixed to the ground and the body frame is fixed to the quadcopter as shown in Figure 1. Figure 1 and the majority of the subsequent equations are taken from [1].

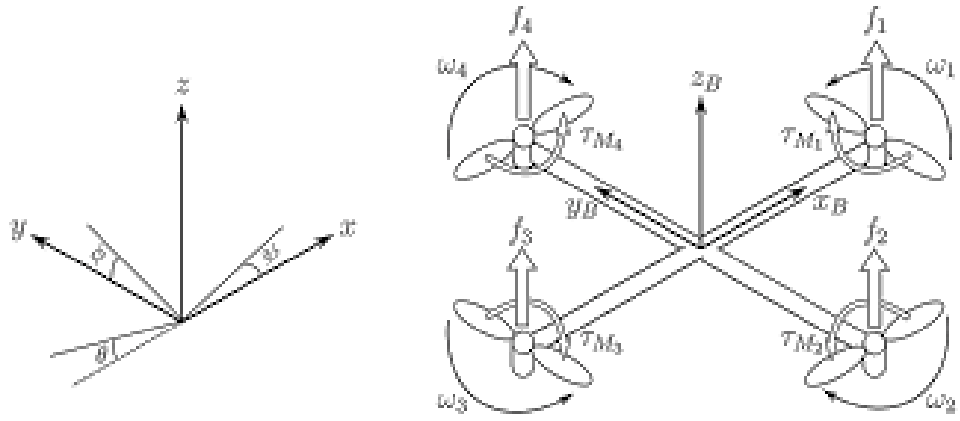


Figure 1: The inertial and body frames.

*Kinematics*

The roll angle, φ, is the rotation about the x-axis, pitch angle, θ, is the rotation about the y-axis, and yaw angle, ψ, is the rotation about the z-axis. The position, velocity, and the attitude of the quadcopter in the inertial frame are defined by equation (1), equation (2), and equation (3) respectively.

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad (1)$$

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} \qquad (2)$$

$$\mathbf{V} = \begin{bmatrix} \varphi \\ \theta \\ \psi \end{bmatrix} \qquad (3)$$

The center of mass of the quadcopter is assumed to be directly in the middle due to symmetry. The linear velocities and the angular velocities in the body frame are defined by equation (4) and equation (5) respectively.

$$\dot{\mathbf{x}}_{\mathbf{B}} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \qquad (4)$$

$$\mathbf{v} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \qquad (5)$$

The direction cosine matrix from the body frame to the inertial frame is shown in equation (6).

$$\mathbf{R} = \begin{bmatrix} C_\theta C_\psi - C_\theta S_\varphi S_\psi & C_\theta S_\psi & -S_\theta \\ S_\theta C_\psi S_\varphi - C_\varphi S_\psi & C_\varphi C_\psi + S_\varphi S_\psi S_\theta & C_\theta S_\varphi \\ S_\varphi S_\psi + C_\varphi C_\psi S_\theta & C_\varphi S_\psi S_\theta - C_\psi S_\varphi & C_\theta C_\varphi \end{bmatrix} \quad (6)$$

This matrix is derived from the 3-2-1 Euler angle transformation matrices. The derivation of **R** can be found in [3]. $\mathbf{W}_V^{-1}$ is used in equation (7) to transform the angular velocities from the body frame to the inertial frame. To transform the angular velocities from the inertial frame to the body frame, the transformation matrix $\mathbf{W}_V$ is used shown in equation (8).

$$\dot{\mathbf{V}} = \mathbf{W}_V^{-1}\mathbf{v}, \begin{bmatrix} \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & S_\varphi T_\theta & C_\varphi T_\theta \\ 0 & C_\varphi & -S_\varphi \\ 0 & S_\varphi / C_\theta & C_\varphi / C_\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \quad (7)$$

$$\mathbf{v} = \mathbf{W}_V\mathbf{V}, \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -S_\theta \\ 0 & C_\varphi & C_\theta S_\varphi \\ 0 & -S_\varphi & C_\theta C_\varphi \end{bmatrix} \begin{bmatrix} \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \quad (8)$$

*Newton-Euler equations*

Newton-Euler equations can be used to describe the dynamics of the quadcopter since it is assumed to be a rigid body. The forces $f_1$, $f_2$, $f_3$, and $f_4$ are the forces from the motor and propeller combination. $\omega_1$, $\omega_2$, $\omega_3$, and $\omega_4$ are the angular velocities of each propeller. The

quadcopter that will be modeled in the simulation is assumed to be symmetric where $I_{xx}$ is equal to $I_{yy}$. So, the inertia matrix is diagonal and shown in equation (9).

$$\mathbf{I} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \qquad (9)$$

The force created by each propeller is shown in equation (10) where k is the lift coefficient.

$$f = k\omega^2 \qquad (10)$$

A torque around the axis of the rotor is created by the angular velocity and acceleration and shown in equation (11) where the inertia moment of the rotor is $I_M$ and the drag constant is b.

$$\tau_M = b\omega^2 + I_M\dot{\omega} \qquad (11)$$

Usually $\dot{\omega}$ is very small and this term is negligible and is assumed to be in this paper. Equations (12), (13), and (14) are the accelerations of the quadcopter in the x, y, and z axis respectively. The pitch acceleration, roll acceleration, and the yaw acceleration are below by equations (15), (16), and (17) respectively. The gravitational constant is g and the angular momentum of each propeller is h.

$$du = vr - wq - g\sin(\theta) \qquad (12)$$

$$dv = wp - ur + g\sin(\varphi)\cos(\theta) \qquad (13)$$

$$dw = uq - vp + g\cos(\varphi)\cos(\theta) - \frac{f_1+f_2+f_3+f_4}{m} \qquad (14)$$

$$dp = (I_{yy} - I_{zz})rq - q(h_1 + h_3 - h_2 - h_4) + \frac{a(f_1-f_3)}{I_{xx}} \qquad (15)$$

$$dq = (I_{zz} - I_{xx})pr + p(h_1 + h_3 - h_2 - h_4) + \frac{a(f_4-f_2)}{I_{yy}} \qquad (16)$$

$$dr = (I_{xx} - I_{yy})pq + \frac{b(f_1+f_3-f_2-f_4)}{I_{zz}} \qquad (17)$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = R^T \begin{bmatrix} u \\ v \\ w \end{bmatrix} \qquad (18)$$

# CHAPTER II

# METHODS

The materials that were used to complete this project included Matlab and Simulink, which were used to create the controller and output the results. The first step was to trim the quadcopter at a hover position. This was done by defining the dynamics and the physical characteristics of the quadcopter in Matlab. A block diagram was then created in Simulink shown in Figure 2 below.

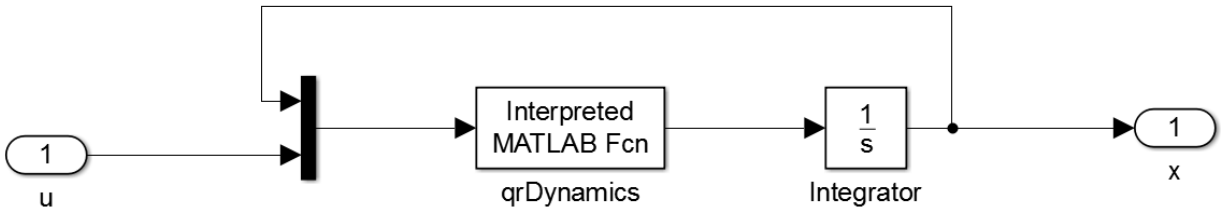

Figure 2: Simulink block diagram

The block diagram has four control inputs which are the forces from the four quadcopter fans. The initial values for the fan forces were chosen to be zero. The initial values for the position of the quadcopter were chosen to be zero. The inputs are used by the ‘qrDynamics‘ function, which is found in the appendix, and then integrated to obtain the twelve states. The quadcopter was then linearized about this equilibrium point to obtain the A,B,C, and D matrices used in equation (20) and equation (21) below.

$$\dot{x} = Ax + Bu \qquad (20)$$

$$y = Cx + Du \qquad (21)$$

Each matrix is a twelve by twelve. The C matrix is the identity matrix and the D matrix is filled with zeros. The portions of the matrices that correspond to the rotational states were then taken which correspond to the bottom right four by four of each matrix.

The next step was to create a controller for all three axis of rotation for the quadcopter. This was done using a modified PID controller to obtain stability. The block diagram for the problem is shown below.

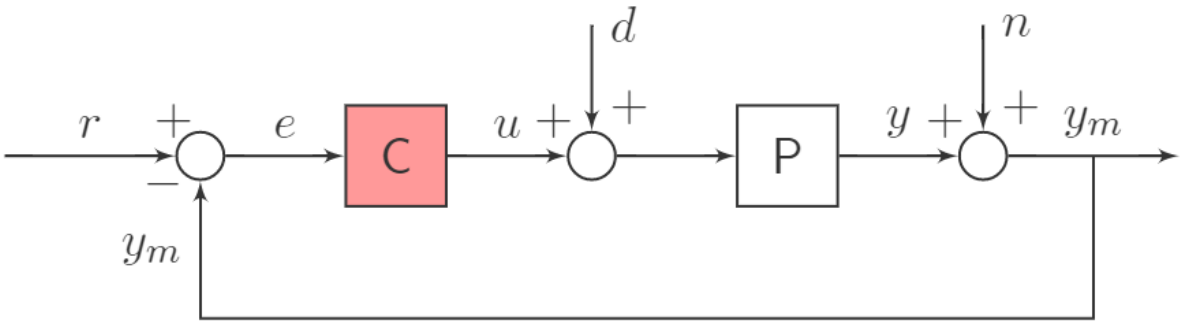


Figure 3: Block diagram for the control problem.

The reference input, r, was taken to be zero since that is the angle the quadcopter should be at for hovering. The noise, n, was also taken to be zero. The 'C' block is the controller. The 'P' block is the plant and is $\frac{1}{Is^2}$, where I is the moment of inertia. The output of the controller is the required moment to restore the quadcopter back to a rotation of zero radians. The signal, after being multiplied by the plant, becomes the angle which is then fed back. Since the plant has two poles at the origin, a zero was added to push the poles to the left hand plane (LHP) for stability. This zero was chosen to be -1.79 and this was chosen by trial and error. Using the root locus method, the gains were chosen by looking at the root locus plot and choosing gains that would

give the desired response, which was an iterative process. The gains chosen to obtain the response shown in the results are $K_P = 0.95, K_D = 0.95$, and $K_I = 4.9$. Once stability was achieved for all three axes, the required moment was plotted versus time. These required moments were then dived by the length of the moment arm, L. This was done to show the fan forces that would be required when using the gains that were picked for the controller.

# CHAPTER III

# RESULTS

The results of the controller that was designed for the quadcopter are presented below. The disturbance rejection to a step response, the required moment needed to produce that step response, and the required fan forces to produce the required moment are all depicted below. All of these depictions are presented for all three axis of rotation. For pitch and roll, the graphs do not differ due to the moments of inertia for those rotations being identical.

## Pitch and Roll Axis

The plot in Figure 4 below shows how the quadcopter responds to a disturbance step response of one radian in the pitch axis and roll axis. The y-axis is in radians while the x-axis is in seconds. The max amplitude that the quadcopter will produce is about 24% of the total input. It takes about 1.7 seconds for the steady-state error to be within 2.5%.

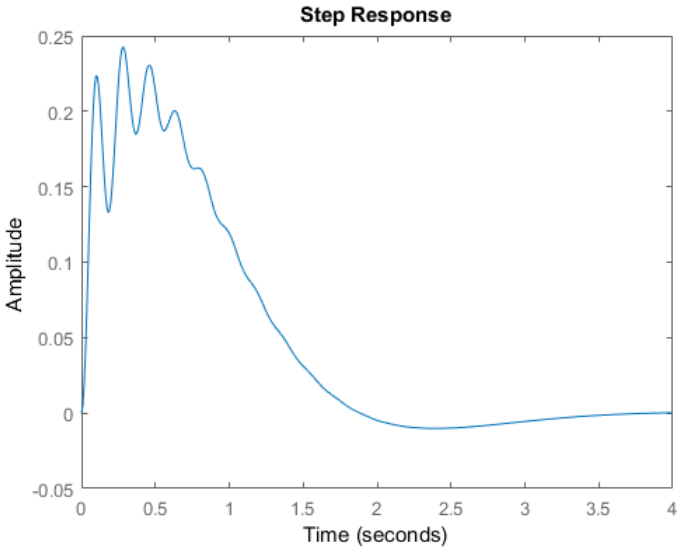


Figure 4: Disturbance response to a step input for the pitch axis.

Figure 5 shows the required moment to produce the response in Figure 4 and is the signal produced by the controller. The y-axis has units of N*m. The moment is negative due to the choice in the positive direction for rotation.

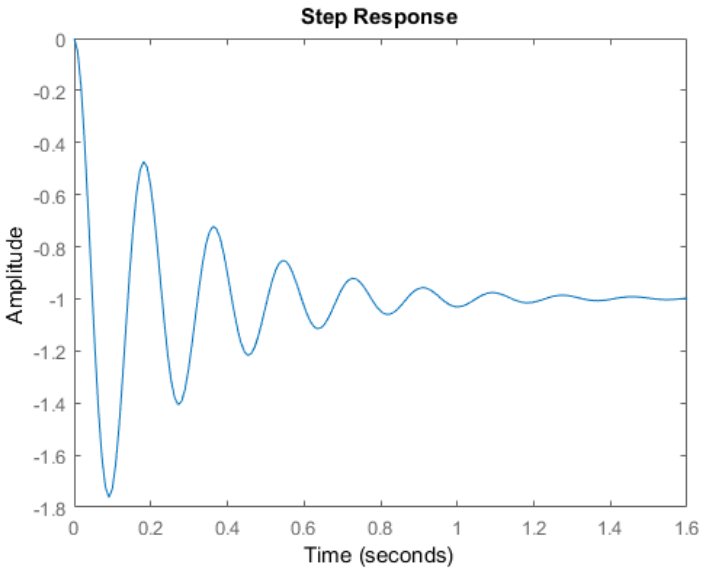


Figure 5: The required moment produced by the controller for the pitch axis.

The required force needed to produce the moment in Figure 5 is shown in Figure 6. This force will be split up between two fans and the max value is about 8.5 N. So, a common electric motor with a recommended propeller could easily produce this force when split between multiple fans.

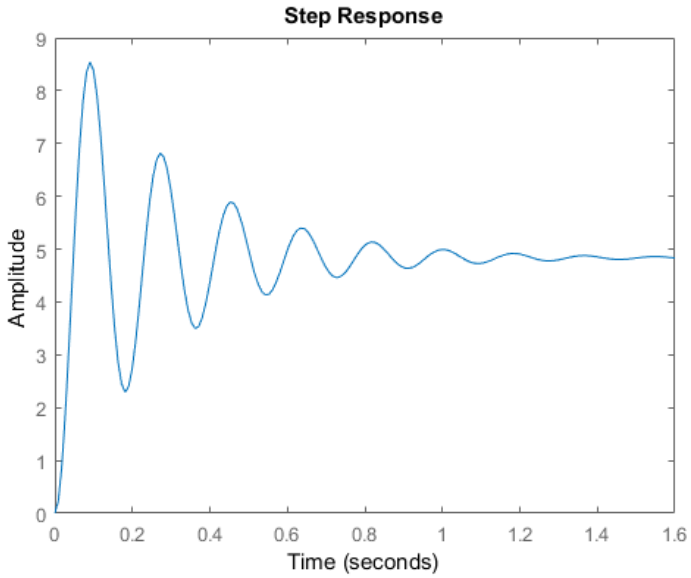

Figure 6: The required force by the fans for the pitch axis.

**Yaw Axis**

The plot in Figure 7 below shows how the quadcopter responds to a disturbance step response of one radian in the yaw axis. The y-axis is in radians while the x-axis is in seconds. The max amplitude that the quadcopter will produce is about 24.5% of the total input. It takes about 1.7 seconds for the steady-state error to be within 2.5%.

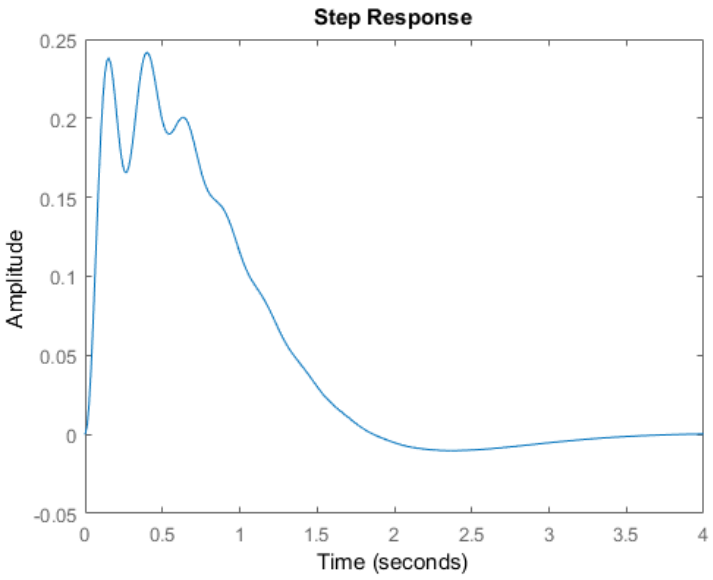


Figure 7: Disturbance response to a step input for the yaw axis.

Figure 8 shows the required moment to produce the response in Figure 7 and is the signal produced by the controller. The y-axis has units of N*m. The moment is negative due to the choice in the positive direction for rotation.

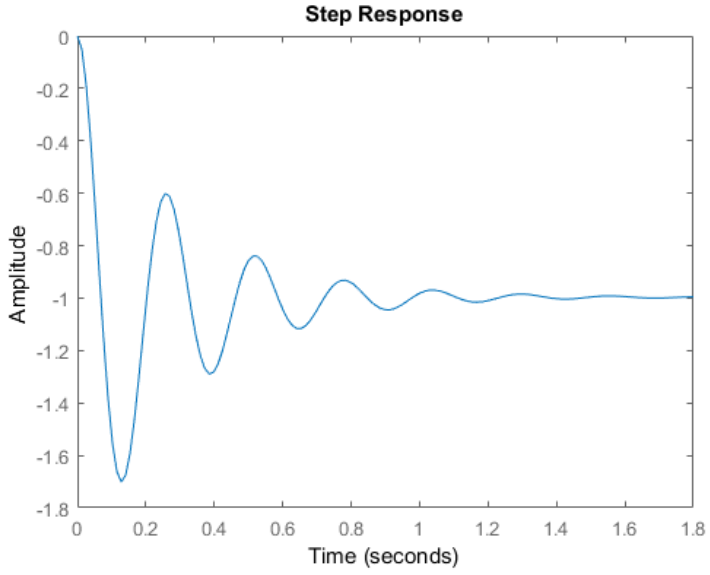


Figure 8: The required moment produced by the controller for the yaw axis.

The required force needed to produce the moment in Figure 8 is shown in Figure 9. This force will be split up between four fans and the max value is about 8.5 N. fans.

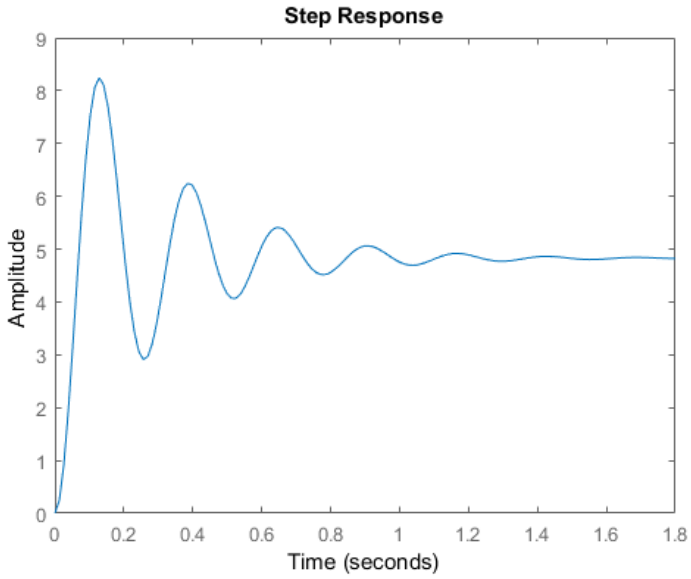


Figure 9: The required force by the fans for the yaw axis.

# CHAPTER IV

# CONCLUSION

The results of this project show that, with the designed controller, that a quadcopter can be stabilized for all six DOFs. The step response is adequate to observe a point using the criterion in [6]. The point could be observed even better if the quadcopter was equipped with a gimbal mounted camera. For future work, a simulation would be made to show how well the quadcopter would track a given trajectory. An algorithm would then be made that would dictate the most optimal trajectories for each quadcopter in the swarm to follow similar to the methods presented in [4], [5], and [8]. A method of determining the distance relative to the point of interest for observation would be needed. This could be done using a camera with an algorithm to determine the distance similar to the methods used in [7].

# REFERENCES


[1] Luukkonen, Teppo. "Modelling and control of quadcopter." *Independent research project in applied mathematics, Espoo* (2011).

[2] Franklin, Gene F., J. David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*. Boston: Pearson, 2015. Print.

[3] Rao, Anil Vithala. *Dynamics of Particles and Rigid Bodies: A Systematic Approach*. Cambridge: Cambridge UP, 2006. Print.

[4] Leonard, Jeremie, Al Savvaris, and Antonios Tsourdos. "Towards a fully autonomous swarm of unmanned aerial vehicles." *Control (CONTROL), 2012 UKACC International Conference on*. IEEE, 2012.

[5] Yu, Bocheng, et al. "Formation control for quadrotor swarm systems: Algorithms and experiments." *Control Conference (CCC), 2013 32nd Chinese*. IEEE, 2013.

[6] Kim, JeongWoon, and David Hyunchul Shim. "A vision-based target tracking control system of a quadrotor by using a tablet computer." *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*. IEEE, 2013.

[7] Ma'sum, M. Anwar, et al. "Autonomous quadcopter swarm robots for object localization and tracking." *Micro-NanoMechatronics and Human Science (MHS), 2013 International Symposium on*. IEEE, 2013.

[8] Vásárhelyi, Gábor, et al. "Outdoor flocking and formation flight with autonomous aerial robots." *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014.

# APPENDIX

## qrDynamics

```
function [ qrStateDerivative ] = qrDynamics(qrStateAndControl)
% QRDYNAMICS Calculates the derivative of a quadrotor state given control
% input

% initialize variables
    qrState = qrStateAndControl(1:12);
    control = qrStateAndControl(13:16);

    % from qrState
    % positions in m; rotations in rad;
    x=qrState(1);
    y=qrState(2);
    z=qrState(3);
    u=qrState(4);
    v=qrState(5);
    w=qrState(6);
    phi=qrState(7);
    theta=qrState(8);
    psi=qrState(9);
    p=qrState(10);
    q=qrState(11);
    r=qrState(12);


   % from fan actuators
    % T in Newtons
    TF = control(1);
    TB = control(2);
    TL = control(3);
    TR = control(4);

    % Drag moment in N-m
    MDF = 0;
    MDB = 0;
    MDL = 0;
    MDR = 0;

    % angular momentum of spinning fan assemblies (kg*m^2/s)
    hF = 0;
    hB = 0;
    hL = 0;
    hR = 0;

    % constants m, a, I's
    % MASS
    m = .537;
    g = 9.8;
```

## qrDynamics Continued

```
    % moment arm for a quad rotor
    a = .2064;

    % drag coefficient
    b = 1;

    % moments of inertia
    Ixx = .0085;
    Iyy = .0085;
    Izz = .0167;

    % attitude influence matrix
    A = [1, sin(phi)*tan(theta), cos(phi)*tan(theta);
    0, cos(phi), -sin(phi);
    0, sin(phi)/cos(theta), cos(phi)/cos(theta)];

    % direction cosine matrix
    C = [                                cos(psi)*cos(theta),
cos(theta)*sin(psi),          -sin(theta);
    cos(psi)*sin(phi)*sin(theta) - cos(phi)*sin(psi), cos(phi)*cos(psi) +
sin(phi)*sin(psi)*sin(theta), cos(theta)*sin(phi);
    sin(phi)*sin(psi) + cos(phi)*cos(psi)*sin(theta),
cos(phi)*sin(psi)*sin(theta) - cos(psi)*sin(phi), cos(phi)*cos(theta)];


du = (v*r-w*q-g*sin(theta));
dv = (w*p-u*r+g*sin(phi)*cos(theta));
dw = (u*q-v*p+g*cos(phi)*cos(theta))-(TF+TB+TL+TR)/m;

dp = ((Iyy-Izz)*r*q-q*(hF+hB-hR-hL)+a*(TL-TR))/Ixx;
dq = ((Izz-Ixx)*p*r+p*(hF+hB-hR-hL)+a*(TF-TB))/Iyy;
dr = ((Ixx-Iyy)*p*q+b*(TF+TB-TL-TR))/Izz;

dPos = C'*[u,v,w]';
dOrientation = A*[p,q,r]';

dphi = dOrientation(1);
dtheta = dOrientation(2);
dpsi = dOrientation(3);

dx = dPos(1);
dy = dPos(2);
dz = dPos(3);

% put derivatives into qrStateDerivative
qrStateDerivative = [dx,dy,dz,du,dv,dw,dphi,dtheta,dpsi,dp,dq,dr]';

end
```

## Trim Quadcopter

```
% Trim quadrotor

clc; clear;
% Trim at hover
X0 = [0 0 0 0 0 0 0 0 0 0 0 0]';
U0 = zeros(4,1);
Y0 = X0;
IX = (1:12)';
IU = [];
IY = IX;

[xbar,ubar,ybar,dxbar] = trim('quadSimulink',X0,U0,Y0,IX,IU,IY);

% Linearize using linmod(....)
[A,B,C,D] = linmod('quadSimulink',xbar,ubar);

Arot = A(7:12,7:12);
Brot = B(7:12,:);

syms phi th psi p q r real
syms F1 F2 F3 F4 real

x = [phi th psi p q r]';
u = [F1 F2 F3 F4]';
xd = Arot*x + Brot*u;
```

## Pitch Controller

```
% PID Control System Design with Root Locus
% ========================================
clear; clc;
s = tf('s');

% 3 Controller structures
beta = 1.79;
P = 1;
I = 1/s;
alp = 10; D = (s+beta)/(s/alp+1);

% Open loop system
Ixx = 0.0085;
G = 1/(Ixx*s^2);

% Check step response of open-loop
figure(2); step(G);

% Default design values
Kp = 0; Ki = 0;Kd = 0;
C = Kp*P + Ki*I + Kd*D;

%% Proportional design
```

## Pitch Controller Continued

```
Lp = G/(1 + G*Kd*D + G*Ki*I);
figure(1);
rlocus(Lp); % Determine Kp from this.
%Kp = 0.7; % Watch rise time
Kp = 0.95; % Watch rise time
% Check step response of closed-loop
C = Kp*P + Ki*I + Kd*D;
figure(2);
step(G/(1+G*C));

%% Derivative design
Ld = G*D/(1 + G*Kp*P + G*Ki*I);
figure(1); rlocus(Ld); % Determine Kd from this.
%Kd = 1;
Kd = 0.95; % Watch overshoot
% Check step response of closed-loop
C = Kp*P + Ki*I + Kd*D;
figure(2);
step(G/(1+G*C));

%% Integral design
Li = G*I/(1 + G*Kp*P + G*Kd*D);
figure(1); rlocus(Li); % Determine Ki from this.
%Ki = 30;
Ki = 4.9; % Watch settling time
% Check step response of closed-loop
C = Kp*P + Ki*I + Kd*D;
figure(2); step(G/(1+G*C));

%% Response to impulse
figure(3);
Gud = -G*C/(1+G*C);
step(Gud)
figure(4);
step(-Gud/0.2064)

% Iterate
```

## Roll Controller

```
% PID Control System Design with Root Locus
% ========================================
clear; clc;
s = tf('s');

% 3 Controller structures
beta = 1.79;
P = 1;
I = 1/s;
alp = 10; D = (s+beta)/(s/alp+1);

% Open loop system
```

## Roll Controller Continued

```
Iyy = 0.0085;
G = 1/(Iyy*s^2);

% Check step response of open-loop
figure(2); step(G);

% Default design values
Kp = 0; Ki = 0;Kd = 0;
C = Kp*P + Ki*I + Kd*D;

%% Proportional design
Lp = G/(1 + G*Kd*D + G*Ki*I);
figure(1);
rlocus(Lp); % Determine Kp from this.
%Kp = 0.7; % Watch rise time
Kp = 0.95; % Watch rise time
% Check step response of closed-loop
C = Kp*P + Ki*I + Kd*D;
figure(2);
step(G/(1+G*C));

%% Derivative design
Ld = G*D/(1 + G*Kp*P + G*Ki*I);
figure(1); rlocus(Ld); % Determine Kd from this.
%Kd = 1;
Kd = 0.95; % Watch overshoot
% Check step response of closed-loop
C = Kp*P + Ki*I + Kd*D;
figure(2);
step(G/(1+G*C));

%% Integral design
Li = G*I/(1 + G*Kp*P + G*Kd*D);
figure(1); rlocus(Li); % Determine Ki from this.
%Ki = 30;
Ki = 4.9; % Watch settling time
% Check step response of closed-loop
C = Kp*P + Ki*I + Kd*D;
figure(2); step(G/(1+G*C));

%% Response to impulse
figure(3);
Gud = -G*C/(1+G*C);
step(Gud)
figure(4);
step(-Gud/0.2064)

% Iterate
```

## Yaw Controller

```
% PID Control System Design with Root Locus
% ========================================
clear; clc;
s = tf('s');

% 3 Controller structures
beta = 1.79;
P = 1;
I = 1/s;
alp = 10; D = (s+beta)/(s/alp+1);

% Open loop system
Izz = 0.0167;
G = 1/(Izz*s^2);

% Check step response of open-loop
figure(2); step(G);

% Default design values
Kp = 0; Ki = 0;Kd = 0;
C = Kp*P + Ki*I + Kd*D;

%% Proportional design
Lp = G/(1 + G*Kd*D + G*Ki*I);
figure(1);
rlocus(Lp); % Determine Kp from this.
%Kp = 0.7; % Watch rise time
Kp = 0.95; % Watch rise time
% Check step response of closed-loop
C = Kp*P + Ki*I + Kd*D;
figure(2);
step(G/(1+G*C));

%% Derivative design
Ld = G*D/(1 + G*Kp*P + G*Ki*I);
figure(1); rlocus(Ld); % Determine Kd from this.
%Kd = 1;
Kd = 0.95; % Watch overshoot
% Check step response of closed-loop
C = Kp*P + Ki*I + Kd*D;
figure(2);
step(G/(1+G*C));

%% Integral design
Li = G*I/(1 + G*Kp*P + G*Kd*D);
figure(1); rlocus(Li); % Determine Ki from this.
%Ki = 30;
Ki = 4.9; % Watch settling time
% Check step response of closed-loop
C = Kp*P + Ki*I + Kd*D;
figure(2); step(G/(1+G*C));

%% Response to impulse
```

```
figure(3);
Gud = -G*C/(1+G*C);
step(Gud)
figure(4);
step(-Gud/0.2064)

% Iterate
```